A[T1]{fontenc}

\usepackage[utf8]{inputenc}

\usepackage{geometry}

\geometry{verbose,tmargin=2cm,bmargin=2cm,lmargin=2cm,rmargin=2cm}

%\usepackage[linesnumbered,ruled,vlined,portuguese]{algorithm2e}

\input{algo}

\RequirePackage{times}

\makeatletter

\newenvironment{code}

{\par\begin{list}{}{

\setlength{\rightmargin}{\leftmargin}

\setlength{\listparindent}{0pt}% needed for AMS classes

\raggedright

\setlength{\itemsep}{0pt}

\setlength{\parsep}{0pt}

\normalfont\ttfamily}%

\item[]}

{\end{list}}

\setlength{\jot}{0.25\jot}

\usepackage{latexsym}

\makeatother

\usepackage{babel}

\usepackage{graphicx}

\title{A project with images}

\author{Overleaf}

\date{}

\begin{document}

\title{Os Anéis de Saturno}

\author{ André Luiz Marques Macrini Leite \thanks{\texttt{andre.marques@edu.pucrs.br}}, Giovanni Frozza \thanks{\texttt{giovanni.frozza@edu.pucrs.br}}\\

Escola Politécnica — PUCRS}

\date{\today}

\maketitle

\begin{abstract}

\emph{O artigo presente irá conduzir à uma das possíveis soluções de como se locomover nos diversos portais ou caminhos encontrados nos Anéis de Saturno. Os Anéis de Saturno é um ambiente que possui vida inteligente que possuem uma rede de portais nas bordas externas e internas do anel (Cada portal é nomeado de de A a Z para a rede externa e também A a Z para a rede interna) que permitem a locomoção dos habitantes. Um habitante do Anel pode se teleportar por portais que contém o mesmo nome ou então caminhando de um portal ao outro. Neste artigo será apresentado um algoritmo que visa encontrar o menor caminho para chegar do portal A ao portal Z}

\end{abstract}

\section{Introdução}

O problema apresentado no Anéis de Saturno requer entender as complexas ligações entre os diversos portais que o Anel possui. O exercício instrui o desenvolvimento de um algoritmo que irá percorrer essas ligações entre a rede de portais e retornar à quantidade de tempo que leva para percorrer do portal A ao portal Z da borda interna do Anel. Cada movimento realizado no Anel é equivalente a uma (1) unidade de tempo e podem ser feitos através dos espaços livres o por uma letra do alfabeto respectivamente.

Para conseguir entender as ligações a leitura do problema apresenta uma lista com as seguintes instruções:

\begin{itemize}

\item A – Z representa os portais;

\item # representa um rohedo por onde não sepode passar;

\item . representa espaço livro

\end{itemize}

A leitura do problema inda explica a NASA apresentou uma foto para entender a estrutura do Anel, a localização dos portais, dos espaços livres e dos obstalos.

Para exemplificar, é apresentada a seguinte estrutura, que se refere ao “caso-teste”:

\vspace{0.3cm}

\begin{figure}[htp]

\centering

\includegraphics{img01.png}

\end{figure}

\vspace{0.3cm}

Os habitantes não podem se mover nas diagonais e para ir de A (externo) a Z (interno) precisando somente de 9 unidades de tempo para se mover entre os portais de partida e destino.

RESULTADO CAMINHAMENTO

A Z

A esta em 3 1

Z esta em 14 26

A 3 1 (0)

. 3 2 (1)

. 2 2 (2)

. 2 3 (3)

. 2 4 (4)

. 2 5 (5)

. 2 6 (6)

K 1 6 (7)

K 14 27 (8)

Z 14 26 (9)

vspace{0.3cm}

Seguindo as regras da rede de portais do Anel foi possível desenvolver um algoritmo para a resolução do problema. Ao final do artigo será apresentado a identificação dos casos de teste contendo seus resultados.

\section{Matrizes, Grafos e BFS}

\subsection{Matriz}

Matrizes são estruturas de dados que apresentam um conjunto de valores do mesmo tipo, partindo do princípio a linguagem de programação Java possui arrays e classes multidimensionais que permite uma série de classes de coleções de objetos. Para a declaração de uma matriz em Java deverão ser fornecidas o tipo dos elementos que serão armazenados na matriz e o número de linhas e colunas que representam a dimensão. Logo, a linguagem Java permite ao programador criar uma estrutura de arrays “unidimensionais” no qual seus elementos recebem outro arrays “unidimensionais”, alcançando assim a mesma estrutura de mesmo efeito que a “multidimensional”.

\subsection{Grafo}

Um grafo é um tipo abstrato de dados permite representar relações entre dois objetos. Os objetos são representados vértices e as relações entre dois objetos por uma aresta. Os vértices do grafo é normalmente representado por círculos ou retângulos e suas relações formam um diagrama que podem ou não ser conectados por arestas que são tipicamente representadas por segmentos de retas ou arcos, com ou sem setas.

\subsection{BFS – Busca em largura}

Para IME-USP (2019)[3] a busca em largura, ou o termo em inglês breadth-first search(BFS), tem o objetivo de visitar a ponta inicial de um arco percorrendo pelos arcos de um vértice a outro, onde cada arco visitado é percorrido no máximo uma vez.

O BFS O algoritmo numera os vértices, em sequência, na ordem em que eles são descobertos (ou seja, visitados pela primeira vez), ou seja, o algoritmo a si próprio e depois todos os seus vizinhos, e depois os vizinhos dos vizinhos, e assim por diante. Através deste algoritmo é possível descobrir o menor caminho de um vértice a outro.

\section{Algoritmo OAS}

O algoritmo OAS (Os Aneis de Saturno) desenvolvido para este desafio utiliza a linguagem de programação JAVA e contém duas classes public, “class App”, “class Coordenadas” e “class Grafo”.

\vspace{0.3cm}

Classe App

- classe App contém o método main() e onde é faz a leitura dos dados de entrada e retorna os resultados com auxílio do método getTamanhoLinha(). Esta classe lê a primeira linha do arquivo de caso de teste e com base no tamanho encontrado, é usado para estanciar a classe Grafo. Depois é realizado toda a leitura do arquivo onde vai ser montado uma matriz com auxilio do método insereLinha da classe Grafo.

Este classe contém um método recebe por parâmetro uma String, um arquivo .txt, e fará a conversão deste arquivo utilizando a interface das classes File, FileReader e BufferReader onde irá introduzir em um arranjo de String que irá ler os primeiros dois valores do arquivo .txt que irá determinar as coordenadas de origem e destino.

\vspace{0.3cm}

Classe Coordenadas

- A classe Coordenadas apenas contém o método construtor que recebe dois parâmetros do tipo inteiro que representam o eixo x e y da matriz. Estes valores são importantes para a construção e mapeamento da matriz, que vão ser utilizados por quase todas as funções do algoritmo para diferentes finalidades.

\vspace{0.3cm}

Classe Grafo

A classe foi pensada inspirada no problema do Passeio do Cavalo, onde a estrutura do grafo é uma matriz.

- Esta classe é composta por 9 variáveis globais onde 4 são matrizes, 2 da do tipo Coordenadas e 3 do tipo int. Além disso é composto também por um construtor que inicializa as matrizes e as variáveis do tipo inteiro.

- O método econtraCaminho() que recebe a origem e destino que são lidas na primeira linha do arquivo de caso de teste. Neste método a primeira coisa a ser feita é definir as coordenadas do portal de origem e o portal de destino pelas variáveis globais Coordenadas portalInicio e portalFim respectivamente. Após definido as coordenadas são chamado o método bfs.

- O método bfs() é possível saber a distância de um vértice para qualquer outro vértice. O método recebe duas coordenadas para descobrir a menor distância entre estas duas coordenadas. O algoritmo bfs utilizado neste trabalho é o que foi apresentado em aula, mas adaptado para usar matriz comum ao invés matriz adjacente como apresentado originalmente.

Neste algoritmo cria-se uma Lista de Coordenadas e instância uma fila. Usa-se uma matriz para sinalizar os marcados, uma matriz para sinalizar da onde a marcação veio e a distância até a última marcação. Com uma estrutura de repetição “while” para que enquanto a fila estiver preenchida ele verifica se os vértices vizinhos já estão marcados ou se estão em branco. Sendo branco ele marca que o vértice foi visitado, regista a origem da marcação, caminha para a próxima coordenada e adiciona na lista.

Se encontrado a coordenada destino o método é encerrado.

\begin{lstlisting}

|  |
| --- |
| private void bfs(int x, int y) { |
|  | List<Coordenadas> q = new LinkedList<>(); |
|  | Coordenadas cood = new Coordenadas(x, y); |
|  | q.add(cood); |
|  | marcados[cood.x][cood.y] = true; |
|  | edgeTo[x][y] = new Coordenadas(-1, -1); |
|  | distTo[x][y] = 0; |
|  | boolean acabou = false; |
|  | while (!q.isEmpty()) { |
|  | Coordenadas v = q.remove(0); |
|  | for (Coordenadas w : this.getAdj(v.x, v.y)) { |
|  | if ( !marcados[w.x][w.y] && !mat[w.x][w.y].equals("#") && !mat[w.x][w.y].isBlank() ) { |
|  | //Coordenadas to = !mat[w.x][w.y].equals(".") ? this.getPortalInterno(mat[w.x][w.y]) : new Coordenadas(w.x, w.y); |
|  | Coordenadas to = new Coordenadas(w.x, w.y); |
|  | marcados[to.x][to.y] = true; |
|  | edgeTo[to.x][to.y] = new Coordenadas(v.x, v.y); |
|  | distTo[to.x][to.y] = distTo[v.x][v.y] + 1; |
|  | q.add(to); |
|  | if (w.x == this.portalFim.x && w.x == this.portalFim.y) acabou = true; |
|  | } |
|  | } |
|  | if (acabou) break; |
|  | } |
|  | } |

\end{lstlisting}

- O método getAdj() faz uma série de verificações responsável por olhar para os vértices vizinhos e para o vértice corrente. Se o vértice corrente for um portal será chamado o método getPortal().

\begin{lstlisting}

|  |
| --- |
| private Coordenadas getPortal(int x, int y) { |
|  | if (x == 0 || x == this.altura - 1 || y == 0 || y == this.largura - 1) { |
|  | return this.getPortalInterno(this.mat[x][y]); |
|  | } else { |
|  | return this.getPortalExterno(this.mat[x][y]); |
|  | } |
|  | } |

\end{lstlisting}

- O método getPortal() faz duas verificações, se o o portal é Externo ou Interno. Dependendo da condição válida ele vai chamar o método correspondente oposto. Por exemplo, se o vértice corrente for um portal Externo, então a codição irá chamar o método getPortalInterno() e vice-versa.

- Os métodos getPortalExterno() e getPortalInterno() são semelhantes em seu funcionamento que contém apenas algumas diferenças de cálculo que determina o tamanho das bordas. Este método contém uma estrutura de repetição “for” e faz duas verificações. Na primeira verificação olha se o portal inicial está em uma das bordas superior ou inferior, se verdadeiro ele usa uma estrutura de repetição para mapear todas as linhas borda superior ou inferior. Se caso a verificação for falsa é criado uma terceira estrutura de repetição “for” para mapear as bordas laterais simultaneamente como uma forma de otimização.

\begin{lstlisting}

|  |
| --- |
| private Coordenadas getPortalInterno(String portal) { |
|  | int x = ( this.altura / 2 ); |
|  | int y = ( this.largura / 2 ); |
|  | int nrDist = 0; |
|  | Coordenadas portalInterno = null; |
|  |  |
|  | for (int i = x; i > 0; i--) { |
|  | if ( !this.mat[i][y].isBlank() ) break; |
|  | nrDist++; |
|  | } |
|  | int posX = ( x - nrDist ); |
|  | int posY = ( y - nrDist ); |
|  | int tam = posX + nrDist \* 2; |
|  | boolean achou = false; |
|  | for (int i = posX; i < tam; i++) { |
|  | if (i == posX || i == tam - 1) { |
|  | for (int j = posY; j < tam; j++) { |
|  | if (this.mat[i][j].equals(portal)) { |
|  | portalInterno = new Coordenadas(i, j); |
|  | achou = true; |
|  | break; |
|  | } |
|  | } |
|  | } else { |
|  | if (this.mat[i][posX].equals(portal)) { |
|  | portalInterno = new Coordenadas(i, posX); |
|  | break; |
|  | } else if (this.mat[i][tam - 1].equals(portal)) { |
|  | portalInterno = new Coordenadas(i, tam - 1); |
|  | break; |
|  | } |
|  | if (achou) break; |
|  | } |
|  | } |
|  | return portalInterno; |
|  | } |

\end{lstlisting}

- O método encontrarCaminho() retorna o “int” que é o custo do caminhamento. O método consiste que após encontrar o vértice destino que pertence a borda interna do Anel, o método bfs para de executar. O método considera que toda a matriz edgeTo e distTo foi preenchida e então o algoritmo consegue olhar a fila e verificar quem veio antes do vértice destino e repetindo o processo sucessivamente até o se encontrar o vértice de origem, mas com auxio do método getCaminho().

\begin{lstlisting}

|  |
| --- |
|  |
| public int encontraCaminho(String valorInicio, String valorFim) { |
|  | Coordenadas coordenadas = this.getPortalExterno(valorInicio); |
|  | this.portalInicio = coordenadas; |
|  | this.portalFim = this.getPortalInterno(valorFim); |
|  | this.bfs(coordenadas.x, coordenadas.y); |
|  | List<Coordenadas> caminho = this.getCaminho(); |
|  | return caminho.size(); |
|  | } |
|  |  |

\end{lstlisting}

- O método getCaminho() vai retornando o caminho do vértice destino até o vértice de origem adicionando-os em uma Lista de Coordenadas. O custo deste caminhamento vai ser o tamanho(.size) da Lista de Coordenadas e vai retornar o método pai, encontrarCaminho().

\begin{lstlisting}

|  |
| --- |
|  |
| |  | | --- | | private List<Coordenadas> getCaminho() { | |  | List<Coordenadas> caminho = new ArrayList<>(); | |  | int x = this.portalFim.x; | |  | int y = this.portalFim.y; | |  | while (x != this.portalInicio.x && y != this.portalInicio.y) { | |  | int auxX = x; | |  | int auxY = y; | |  | x = this.edgeTo[auxX][auxY].x; | |  | y = this.edgeTo[auxX][auxY].y; | |  | caminho.add(new Coordenadas(x, y)); | |  | } | |  | return caminho; | |  | } | |  |

\end{lstlisting}

\vspace{0.3cm}

Este algoritmo determina qual o menor custo possível para fazer o caminhamento pela rede de portais nos Anéis de Saturno. A notação deste algoritmo pode se dar em seu melhor caso O(1) e no pior caso O(n²), pois devemos considerar que o Anel pode induzir a visitar todas as posições permitidas da matriz para chegar do portal A ao portal Z.

\section{Estudo de casos}

O exercício consiste em apresentar os resultados dos oito estudos de casos, além do caso de teste que foi exemplificado na introdução do artigo. Segue abaixo a tabela contendo a resposta esperada e obtida para cada caso:

RESULTADO ESPERADO E RESULTADO OBTIDO

\begin{figure}[htp]

\centering

\includegraphics{resultadoCasos.png}

\end{figure}

\section{Conclusão}

O desafio apresento grau de dificuldade médio e a resolução poderia ser desenvolvida usando até mesmo leitura simples de arranjos que resultaria uma notação de O(n³). Mas para o desafio proposto foi compreendido que utilizando apenas uma matriz simples seria o suficiente para solucionar o esquema complexo da rede de portais dos Anéis de Saturno.

O algoritmo implementado utilizou algumas noções grafos, como por exemplo vértices, arestas e o funcionamento sobre busca em largura, no qual foi implementado, para a construção e caminhamento de estruturas de grafos. O algoritmo processou as informações e obteve um resultado aproximado do esperado, logo, mostrando que o algoritmo atingiu parcialmente as expectativas e chegou próximo do proposto pelo desafio.

\newpage

\section{Referências}

\begin{thebibliography}{3}

\bibitem{texbook}

DEVMEDIA, Matrizes - Aprenda a trabalhar com vetores bidimensionais - Revista easy Java Magazine 22. Disponível em: <https://www.devmedia.com.br/matrizes-aprenda-a-trabalhar-com-vetores-bidimensionais-revista-easy-java-magazine-22/25766> Acessado em 21 de novembro de 2020 20:30.

\bibitem{texbook}

DEVMEDIA, Programando com Grafos - Revista easy Java Magazine 29. Disponível em: <https://www.devmedia.com.br/programando-com-grafos-revista-easy-java-magazine-29/28119> Acessado em 21 de novembro de 2020 21:34.

\bibitem{texbook}

IME-USP, Busca em largura. Instituto de Matemática e Estática da Universidade de São Paulo. Disponível em: < https://www.ime.usp.br/~pf/algoritmos\_para\_grafos/aulas/bfs.html> Acessado em 21 de novembro de 2020 21:52.

\end{thebibliography}

\end{document}